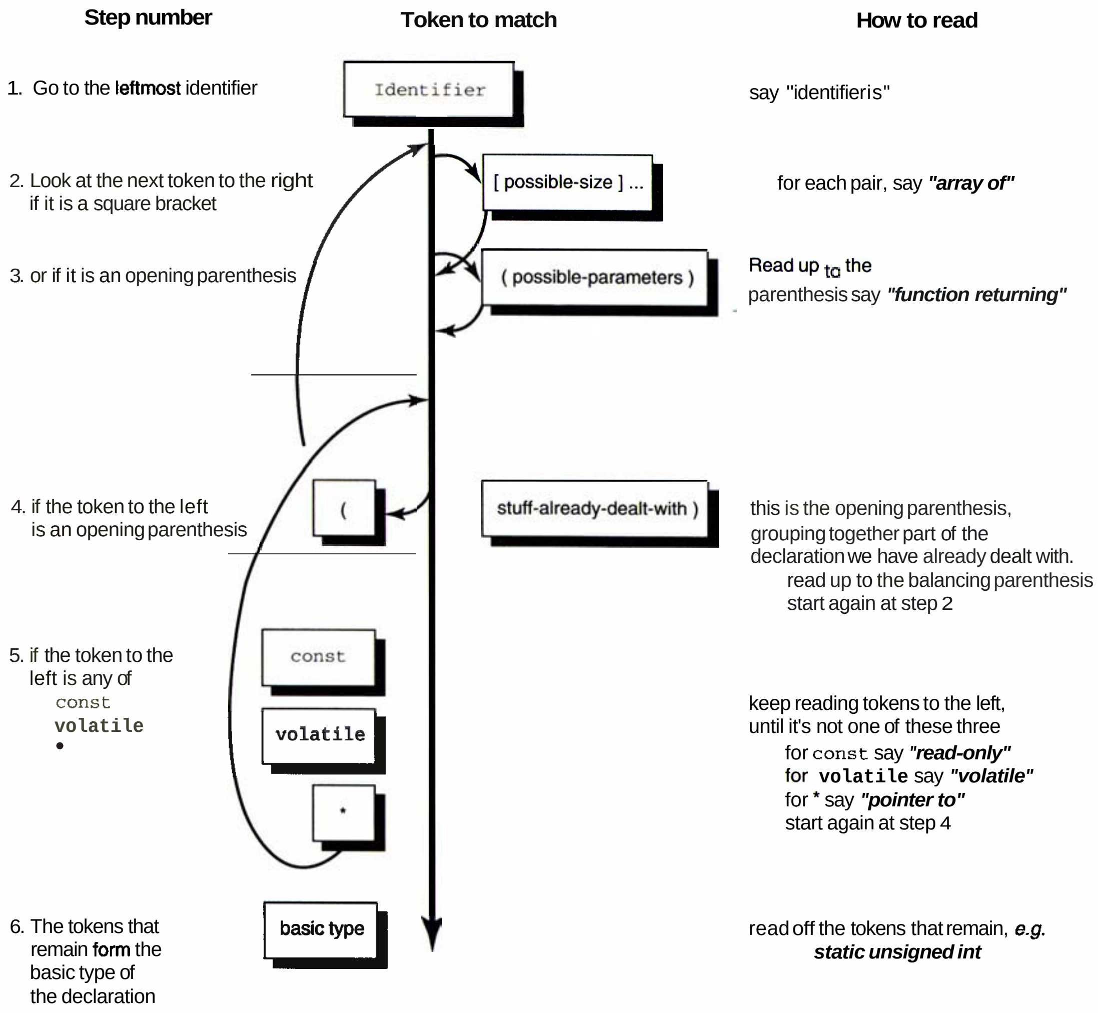
Magic Decoder Ring for C Declarations

Declarations in C are read boustrophedonically, i.e. alternating right-to-left with left-to right. And who'd have thought there would be a special word to describe that! Start at the first identifier you find when reading from the left. When we match a token in our declaration against the diagram, we erase it from further consideration. At each point we look first at the token to the right, then to the left. When everything has been erased, the job is done.



1. Go to the leftmost identifier — say "identifier is"

2. Look at the next token to the right if it is a square bracket — for each pair, say ***"array of"***

3. or if it is an opening parenthesis — Read up to the parenthesis say ***"function returning"***

4. if the token to the left is an opening parenthesis — this is the opening parenthesis, grouping together part of the declaration we have already dealt with.
   read up to the balancing parenthesis
   start again at step 2

5. if the token to the left is any of
   `const`
   `volatile`
   •

   keep reading tokens to the left, until it's not one of these three
   for `const` say ***"read-only"***
   for `volatile` say ***"volatile"***
   for * say ***"pointer to"***
   start again at step 4

6. The tokens that remain form the basic type of the declaration — read off the tokens that remain, ***e.g.*** ***static unsigned int***

***Figure* 3-3** *How to **Parse** a C Declaration*

Let's try a couple of examples of unscrambling a declaration using the diagram. Say we want to figure out what our first example of code means:

```
char* const *(*next)();
```

As we unscramble this declaration, we gradually "white out" the pieces of it that we have already dealt with, so that we can see exactly how much remains. Again, remember `const` means "read-only". Just because it says constant, it doesn't necessarily mean constant.

The process is represented in Table 3-2. In each step, the portion of the declaration we are dealing with is printed in bold type. Starting at step one, we will proceed through these steps.

Table 3-2 Steps in Unscrambling a C Declaration

Declaration Remaining *(start at leftmost identifier)*	**Next Step to Apply**	**Result**
`char * const *(`**`*next`**`) ();`	step 1	say "**next is a...**"
`char * const *(*` **`) ();`**	step 2, 3	doesn't match, go to next step, say "next is a..."
`char * const *(*     ) ();`	step 4	doesn't match, go to next step
`char * const *(*     ) ();`	step 5	asterisk matches, say "**pointer to ...**", go to step 4
`char * const *(      ) ();`	step 4	"(" matches up to ")", go to step 2
`char * const *` **`();`**	step 2	doesn't match, go to next step
`char * const *` **`();`**	step 3	say "**function returning...**"
`char * const *      ;`	step 4	doesn't match, go to next step
`char * const *      ;`	step 5	say "**pointer to...**"
`char *` **`const`** `;`	step 5	say "**read-only...**"
`char *              ;`	step 5	say "**pointer to...**"
`char` `;`	step 6	say "**char**"

Then put it all together to read:

"next is a pointer to a function returning a pointer to a read-only pointer-to-char"

and we're done.